(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 December 2003 (18.12.2003)

PCT

(10) International Publication Number
WO 03/104974 A2

(51) International Patent Classification[7]: G06F 9/00

(21) International Application Number: PCT/US03/11419

(22) International Filing Date: 11 April 2003 (11.04.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/386,856 7 June 2002 (07.06.2002) US
10/165,160 7 June 2002 (07.06.2002) US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 4150 Network Circle, Santa Clara, CA 95054 (US).

(72) Inventor: MONTEMAYOR, Oscar; 1035 Aster Ave. Apt. 1134, Sunnyvale, CA 94086 (US).

(74) Agent: PARK, Richard; 508 Second St., Ste. 201, Davis, CA 95616 (US).

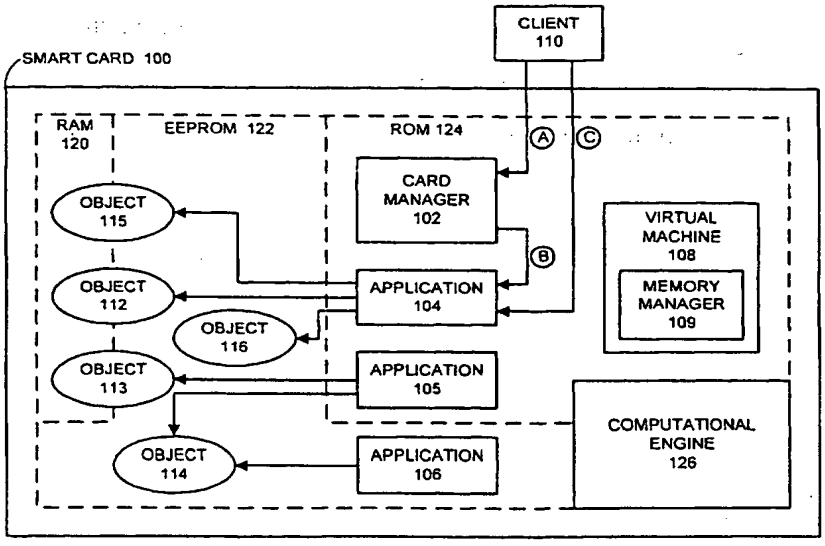
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: USING SHORT REFERENCES TO ACCESS PROGRAM ELEMENTS IN A LARGE ADDRESS SPACE



(57) Abstract: One embodiment of the present invention provides a system that accesses a desired element during execution of a program. During operation, the system receives a reference to the desired element. The system determines if the reference is an internal reference to a location in a local package that is currently executing, or an external reference to a location in an external package. If the reference is an external reference, the system uses an index component of the reference to lookup an address for the desired element in a global reference table. Next, the system uses the address to access the desired element. Note that the address retrieved from the global reference table is larger than the reference, which allows the address to access a larger address space than is possible to access with the reference alone.

WO 03/104974 A2

USING SHORT REFERENCES TO ACCESS PROGRAM ELEMENTS IN A LARGE ADDRESS SPACE

Inventor: Oscar A. Montemayor

BACKGROUND

Field of the Invention

The present invention relates to addressing mechanisms within computer systems. More specifically, the present invention relates to a method and an apparatus for using short references to access program elements in a large address space.

Related Art

Dramatic advances in computer technology presently make it possible to integrate a significant amount of computing power onto "smart cards." Smart cards are used in a variety of applications to solve common security and identity needs. For example, smart cards have been integrated into credit cards, debit cards, corporate badges, and even cell phones.

In order to provide portability for smart card applications, some designers have begun to integrate virtual machines, such as the JAVA virtual machine, into smart cards. Smart cards containing a JAVA virtual machine are often referred to as "JAVA CARDS." Integrating a virtual machine into a smart card enables the smart card execute a large number of platform-independent applications. Moreover, the

associated development environment for the virtual machine can simplify the process of developing applications for smart cards.

Smart card applications are beginning to access large amounts of data. For example, applications that use biometric data, digital certificates and GSM SIM cards must be able to access large amounts of application data, and these requirements are expected to grow ten-fold by the near future.

Unfortunately, the JAVA CARD virtual machine was originally designed to support 16-bit addresses, which limits the virtual machine to accessing 64 kilobytes of memory, unless more sophisticated strategies are used. Hence, existing JAVA CARD systems are likely to be hindered by the lack of storage capacity as well as addressing limitations.

16-bit computer systems typically use segmentation strategies to access memory beyond 64 kilobytes. Segmentation requires complex memory management structures, which consume a large amount of space. This is a problem in memory-constrained devices, such as smart cards, where memory space is very scarce. Furthermore, segmentation requires pre-allocation of memory, which tends to cause fragmentation of the address space, which also results in memory space being used inefficiently.

Hence, what is needed is a method and an apparatus that allows smart card applications to access a larger address space without the above-described disadvantages of segmentation.

SUMMARY

One embodiment of the present invention provides a system that accesses a desired element during execution of a program. During operation, the system receives a reference to the desired element. The system determines if the reference is an internal reference to a location in a local package that is currently executing, or an external reference to a location in an external package. If the reference is an external reference, the system uses an index component of the reference to lookup an address

for the desired element in a global reference table. Next, the system uses the address to access the desired element. Note that the address retrieved from the global reference table is larger than the reference, which allows the address to access a larger address space than is possible to access with the reference alone.

In a variation on this embodiment, if the reference is an internal reference, the system determines a base address of a component in which the desired element is located, and adds an offset of the reference to the base address to produce the address for the desired element.

In a further variation, determining the base address of the component involves obtaining an identifier for the local package that is currently executing, and using the identifier to lookup an entry in a package location table, wherein the entry points to a component location table for the local package. Next, the system examines the component location table to determine the base address of the component. Note that the system may examine an instruction preceding the reference to determine which component the desired element is located in.

In a variation on this embodiment, the system installs a new package by calculating element addresses for all exported references associated with the new package, and then adding the element addresses to the global reference table. Next, the system replaces all external references within the new package with references to entries in the global reference table, wherein the entries in the global reference table contain element addresses for the desired elements.

In a variation on this embodiment, determining if the reference is an internal reference or an external reference involves examining a reserved bit within the reference, wherein the reserved bit indicates whether the reference is an internal reference or an external reference.

In a variation on this embodiment, looking up the address for the desired element in the global reference table involves retrieving an absolute address for the desired element from the global reference table.

In a variation on this embodiment, looking up the address for the desired element in the global reference table involves retrieving a relative address for the desired element from the global reference table, wherein the relative address specifies an offset relative to a base address of a component containing the desired element.

In a variation on this embodiment, the desired element can include: a class record; a static field; or a method.

In a variation on this embodiment, the reference is 16 bits in size and the address retrieved from the global reference table is 32 bits in size.

In a variation on this embodiment, the global reference table is stored as a multi-level table so that the entire global reference table does not have to be allocated at once.

In a variation on this embodiment, a given package includes: a static field component containing static fields; a class component containing class records; a method component containing methods; an export component; and an import component.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1 illustrates a smart card in accordance with an embodiment of the present invention.

FIG. 2 illustrates data structures involved in the addressing operations in accordance with an embodiment of the present invention.

FIG. 3 illustrates a multi-level global reference table in accordance with an embodiment of the present invention.

FIG. 4 illustrates two reference formats in accordance with an embodiment of the present invention.

FIG. 5 is a flow chart illustrating the process of installing a package in accordance with an embodiment of the present invention.

FIG. 6 is a flow chart illustrating the process of converting a reference into an address of a desired element in accordance with an embodiment of the present invention.

FIG. 7 is a flow chart illustrating the process of converting an object reference into an object address in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION

The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

The data structures and code described in this detailed description are typically stored on a computer readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. This includes, but is not limited to, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs) and DVDs (digital versatile discs or digital video discs), and computer instruction signals embodied in a transmission medium (with or without a carrier wave upon which the signals are modulated). For example, the transmission medium may include a communications network, such as the Internet.

Smart Card

FIG. 1 illustrates a smart card 100 in accordance with an embodiment of the present invention. Smart card 100 can generally include any type of miniature computing device, such as may be located within identification cards, client loyalty cards, electronic wallets and cellular telephones. However, note that the present

invention is not meant to be limited to smart cards, and can generally be applied to any type of computing device or computer system that stores objects in writeable non-volatile memory and/or volatile memory.

Smart card 100 contains a computational engine 126, which includes circuitry for performing computational operations. Smart card 100 also contains a number of different types of memory, including random access memory (RAM) 120, electrically erasable programmable read-only memory (EEPROM) 122 and read-only memory (ROM) 124. In general, RAM 120 can include any type of volatile random access memory; EEPROM 122 can include any type of writeable non-volatile memory, such as EEPROM, flash memory, or magnetic memory; and ROM 124 can include any type of read-only memory.

RAM 120 is used to store various data items and data structures. For example, as illustrated in FIG. 1, RAM 120 can contain portions of objects 112, 115 and 113. Note that objects 112, 113 and 115 are "transient objects" that include a persistent portion stored in EEPROM 122, and a transient portion stored in a RAM 120.

ROM 124 includes a virtual machine 108, such as the JAVA virtual machine developed by SUN Microsystems, Inc. of Santa Clara, California. Note that applications written in a platform-independent programming language, such as the JAVA programming language, can be executed on virtual machine 108. Virtual machine 108 includes a memory manager 109 that manages allocation and freeing of memory within smart card 100.

ROM 124 also contains a number of applications, 104 and 105, which provide services for clients accessing smart card 100. Other applications, such as application 106, can be located in EEPROM 122. Further applications (not illustrated) may be located in both ROM 124 and EEPROM 122.

ROM 124 also includes a card manager 102, which contains code for managing the execution of applications on smart card 100. For example, suppose a client 110 wishes to access a service provided by one of the applications 104-106 on smart card 100. Client 110 first communicates with card manager 102 (step A). Card

manager 102 puts client 110 in contact with an application 104 (step B). This allows client 110 to communicate directly with application 104 (step C). Note that card manager 102 can also delete objects from memory.

RAM 120 and EEPROM 122 contain a number of objects 112-116, which are accessed by applications 104-105. More specifically, application 104 accesses objects 112, 115 and 116, application 105 accesses objects 113 and 114, and application 106 accesses object 114.

Data Structures

FIG. 2 illustrates various data structures involved in the addressing operations in accordance with an embodiment of the present invention. These data structures include package location table 202, global reference table 212 and object manager 236. During program execution, these data structures are operated on by an execution engine 200 that is part of virtual machine 108 illustrated in FIG. 1.

FIG. 2 also illustrates a new package 224, which is installed into the system by installer 222. This installation process is described in more detail below with reference to FIG. 5. Package location table (PLT) 202 stores pointers to specific packages and is indexed by a package identifier. For example, the first entry in PLT 202 is associated with a specific package and points to a component location table (CLT) 204 for the specific package. Component location table 204 stores entries for components that make up the specific package. In particular, component location table 204 contains static field component entry 205, class component entry 206, method component entry 206 and export component entry 208. Each of these entries contains a location (base address) for the component as well as the size of the component. For example, static field component entry 205 contains a location 209 for the static field component as well as the size 210 of the static field component. Note that location 209 points to the base address of static field component which contains component data 211. Also note that the import component is not tracked because the

import component is eliminated after it is used during installation as is described below with reference to FIG. 5.

Global reference table (GRT) 212 contains pointers to program elements, such as class records, static fields and methods. Each entry in GRT 212 contains an address 214 of a program element and a package identifier 216, which identifies the package to which the program element belongs. GRT 212 is used to convert short 16-bit external references into longer 32-bit addresses as is described below with reference to FIG. 6.

Object manager 236 is a table that is used to convert object identifiers into object addresses as is described below with reference to FIG. 7.

Multi-Level Global Reference Table

FIG. 3 illustrates a multi-level implementation of global reference table (GRT) 212 in accordance with an embodiment of the present invention. In this embodiment, GRT 212 is divided into two or more levels of indirection, so that the entire table does not have to be allocated at once. The high-level table 304 is allocated the first time the card runs. Elements are subsequently allocated into the respective sub-tables.

When new entries are added into GRT 212, if a lower-level table already allocated exists with empty entries, the entries are added there first by looking for holes (from deleted entries) and then by appending new entries into the table.

When more entries are needed, a new lower-level table is allocated, containing a specified number of entries, which initially are empty. When a low-level table is emptied out due to deletion of all elements, it can be garbage-collected by the virtual machine. Each table is indexed by specific bits of the index reference. For example, in FIG. 3, GRT 212 is divided into 3 levels: a high-level table 304 containing 8 entries (3 bits, indexed by bits 12-14), that are either empty or holding a reference into a second-level table. Each second-level table, such as second-level table 306, can hold

64 entries (6 bits, indexed by bits 6-11 of the reference index), wherein each entry is either empty or contains a reference to a third-level table.

Each third-level table, such as third-level table 308, can hold 64 entries (6 bits, indexed by bits 0-5 of the index reference), and can contain the actual element addresses, or possibly empty entries with no element associated with them.

This indexed design allows for fast access during execution because each index reference can be divided into its bit components that are used to quickly index the table and find the entry. The cost of finding the entry address is: one memory access per level, one shift right operation per level, one multiplication per level, and one addition per level. For the three-level table illustrated in FIG. 3, accesses require 12 operations. These operations are fast, and hence should not cause significant delays during program execution.

Reference Formats

FIG. 4 illustrates two reference formats in accordance with an embodiment of the present invention. As is illustrated in FIG. 4, if the high-order bit of a 16-bit reference is a zero, the remaining 15 bits specify an offset into a local package component. Otherwise, if the high-order bit is a one, the remaining 15 bits specify an index into global reference table 212 illustrated in FIG. 2. This index is used to retrieve the address of a desired element from global reference table 212 as is described below with reference to FIG. 6.

Address Conversion Process

In one embodiment of the present invention, smart card 100 is a JAVA CARD. The JAVA CARD application unit is referred to as a "package." A package contains a set of classes, which could also act as library classes, and can also contain applets, each representing a unique card application. Each package is individually contained inside a Converted APplet (CAP) file--a portable binary format used to distribute JAVA CARD applications. One or more package CAP files can be installed into a

JAVA CARD device at a given time, wherein each package CAP file contains one or more applications.

Packages can interact with each other--normally by performing API calls or by accessing classes, fields or methods defined on different packages. Code inside a given package uses references to access both internal and external elements. These references usually map to the memory address where the desired element is located. Note that a package can export its elements to make them available to other packages to link to.

One embodiment of the present invention relies in the properties of JAVA CARD packages to overcome the 64-kilobyte memory limit for JAVA CARD. When the package is stored in a CAP file, all internal references to packages are given as offsets, while external references are usually associated with tokens that must be mapped against external packages' export components for linking purposes.

In prior JAVA CARD implementations, when a package is installed, all references--both external and internal--are resolved into 16-bit addresses. Hence, in prior implementations a package cannot address memory beyond 64 kilobytes. If the package were to be installed into a 32-bit smart card using a similar technique (where references are represented as 32-bit addresses), all references in the smart card would have to be recalculated, because the CAP file structure implicitly assumes a 16-bit architecture.

One embodiment of the present invention presumes references in the CAP file will remain 16 bits in size, in order to avoid conflict with existing CAP file structures. This embodiment establishes a mapping between the 16-bit references and the actual 32-bit addresses they represent. Note that the references themselves are structured to encode information on how to locate the element address they represent.

All references, upon installation, are modified by the installer 222. Internal references within the package are largely left intact from the CAP file. The high-order bit (bit 15) is marked as "0," while the 15 low-order bits (bits 0-14) encode the offset of the element within the component. Note that a component could be a method, a

class, or a static field. The bytecode that precedes a reference determines how the reference is interpreted.

In order to convert such a reference into an address, the JAVA CARD virtual machine must first know which package it is executing. Once it has this information, it uses the package identifier to obtain a reference to a table containing the individual package's different component addresses. Depending on the bytecode, the lower 15 bits of the reference could be interpreted as an offset into the class, method or static field components of the package. The virtual machine then retrieves from the component table the base address of the component in question and adds the offset from the reference to the address obtained. This produces the memory address of the element desired, which might be at a memory space beyond 16-bit range (i.e. above 64 kilobytes of RAM).

External references, on the other hand, are more complicated to process. During installation, installer 222 revises all references to be exported. For each exported reference, installer 222 calculates the element address (which could be larger than 16 bits) and adds the element address to global reference table 212. (Note that there can be a single global reference table for all elements, or one for each type of element: static fields, methods and classes.) The table structure returns a reference index to installer 222. Installer 222 places this reference index inside the new package's export component. In doing so, the system replaces the referenced entry's package offset in the respective component with the returned reference index. In this way, a package is prepared so that other packages can be linked to it.

In a similar way, all the references that this package makes to other packages are resolved. When an external reference is found, the installer consults the new package's import component and locates the external package that the remote reference is referring to. The installer then goes to the external package's export component (which is already be installed in the card) and retrieves the index reference stored in the export component. Finally, the external reference is replaced by the index reference obtained. An index reference can be differentiated from a local

reference, since the high-order bit (bit 15) is set to "1," while the lower 15 bits (bits 0-14) represent the index into global reference table 212. In addition to storing references, global reference table 212 also stores the package ID associated with the respective reference. Such entry can be used for complex bytecode execution (like checkcast, instanceof), virtual method calls (to find out new package ID of execution) or for package deletion purposes.

Please note that when following the above strategy, the address space that can be used by each component size is limited to 32,768 bytes, a limit imposed by the CAP file format that cannot be expanded. However, this strategy guarantees that each package component will have up to 32,768 bytes for code individually. Hence, each package's static field, class, method and export component can be as long as 32,768 bytes each. However, since each package component can grow up to 32,768 bytes, the addition of all components from all packages can well exceed 32,768 bytes, and even 65,536 bytes (64 kilobytes) in total, because the size of each individual component does not restrict the size of other components. Therefore, the above-described technique overcomes the 64-kilobyte addressing limit.

During bytecode execution, the JAVA CARD virtual machine keeps track of the package where the current code is located. When a local reference is found, the virtual machine consults the local package's component location table to get the component base address and simply adds the offset component from the reference. When the JAVA CARD virtual machine encounters an external reference, it extracts the index component from the reference and uses it to access the respective global reference table. An address, which could be greater than 16 bits in size, is returned. The virtual machine then uses the returned address to access the desired element. In addition, for some bytecodes like checkcast, instanceof, and for all method calls across packages, the JAVA CARD virtual machine can obtain the package ID associated with the element, which is also stored in the global reference table.

Furthermore, each package can create objects, which can be stored at any address, even higher than 64 kilobytes. Newly created objects are inserted into the

object manager data structure, which stores their address (which could be 32 bits in length) and assigns each object a 16-bit reference. Such 16-bit references are stored into static fields or given as parameters for virtual or interface method calls.

Note that the structure of the object manager table 236 in FIG. 2 is similar to global reference table 212. It is merely specialized for objects with optimizations to aid in garbage collection.

For package deletion, the fact that each entry has also a package ID associated with it makes it easier to remove entries from the table. During the package deletion process, two methods can be used: all table elements can be scanned and deleted entries marked, or the export component of the package to delete could be scanned for exported references, and such references marked in the table. Since the table is designed to use holes from deleted entries first, it is guaranteed that slots previously occupied by deleted references will be filled by new references when new packages are installed.

Note that by using the method above on a 16-bit JAVA CARD, a maximum of 32,767 exported elements of each type (objects, methods, static fields, classes) will be able to exist in the card. It is not be possible to have more elements of each type than the available number of index references, since each index reference must be unique inside the smart card.

The different processes involved in the address resolution process are described in more detail below.

Process of Installing a Package

FIG. 5 is a flow chart illustrating the process of installing a new package 224 into a smart card 100 in accordance with an embodiment of the present invention. The system starts by copying the CAP file containing the new package 224 into the smart card 100 (step 502). Next, the system parses the various components of the new package 224, including the class component and the method component (step 504).

During this process, the system leaves local references intact (step 506).

In contrast, imported references are modified. This can involve retrieving the package ID and token ID for the imported reference from the import component of new package 224. Next, the system uses the package ID to locate the export component of the linking package. The system then retrieves a global reference table index from the export component of the linking package (step 508).

Next, the system parses the export component of the new package (step 510). For each entry in the export component, the system resolves the address, adds the address and package ID into the global reference table, and replaces the value in the export component of new package 224 with the index of the address in global reference table 212 (step 512). This allows subsequent packages to link into the export component of new package 224.

Process of Converting an Element Reference into an Element Address

FIG. 6 is a flow chart illustrating the process of converting an element reference into an element address in accordance with an embodiment of the present invention. The process starts by reading a reference (step 602) and then examining the high-order bit of the reference to determine whether the reference is imported or local (step 604).

If the reference is imported, the system uses the index from the reference to lookup the address of the desired element in the global reference table 212 (step 616). The system uses this address to access the desired element (step 614).

If the reference is local at step 606, the system obtains an identifier for the local package that is currently executing (step 606). (This may involve examining the stack.) The system uses this identifier to lookup an entry in package location table 202, which points to a component location table for the package (step 608).

Next, the system gets the base address of the component from the component location table (step 610). This may involve examining a preceding byte code instruction to determine which component the access is directed to.

Next, the system adds the offset to the base address to produce the address of the desired element (step 612). Note that this address can be larger than the original reference. The system then uses the address to access the desired element (step 614).

Process of Converting an Object Reference into an Object Address

FIG. 7 is a flow chart illustrating the process of converting an object reference into an object address in accordance with an embodiment of the present invention. The system starts by obtaining an object reference from the system stack (step 702). The system uses this object to reference and retrieve an entry from the object manager table 236 from FIG. 2 (step 704). If the object is a transient object, the system reads the object header to obtain the location of the object data step (706). The system then adds an element offset to the address of the object to get an address of an element within the object (step 708).

The foregoing descriptions of embodiments of the present invention have been presented only for purposes of illustration and description. They are not intended to be exhaustive or to limit the present invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the above disclosure is not intended to limit the present invention. The scope of the present invention is defined by the appended claims.

What Is Claimed Is:

1. A method for accessing a desired element during execution of a program, comprising:

receiving a reference to the desired element during execution of the program;

determining if the reference is an internal reference to a location in a local package that is currently executing, or an external reference to a location in an external package;

if the reference is an external reference, using an index component from the reference to lookup an address for the desired element in a global reference table; and

using the address to access the desired element;

wherein the address retrieved from the global reference table is larger than the reference, thereby allowing the address to access a larger address space than is possible to access with the reference alone.

2. The method of claim 1, wherein if the reference is an internal reference, the method further comprises:

determining a base address of a component in which the desired element is located; and

adding an offset of the reference to the base address to produce the address for the desired element.

3. The method of claim 2, wherein determining the base address of the component involves:

obtaining an identifier for the local package that is currently executing;

using the identifier to lookup an entry in a package location table, wherein the entry points to a component location table for the local package; and

examining the component location table to determine the base address of the component.

4. The method of claim 3, wherein examining the component location table involves first examining an instruction preceding the reference to determine which component the desired element is located in.

5. The method of claim 1, further comprising installing a new package, wherein installing the new package involves:

calculating element addresses for all exported references associated with the new package;

adding the element addresses to the global reference table; and

replacing all external references within the new package with references to entries in the global reference table, wherein the entries in the global reference table contain element addresses for the desired elements.

6. The method of claim 1, wherein determining if the reference is an internal reference or an external reference involves examining a reserved bit within the reference, wherein the reserved bit indicates whether the reference is an internal reference or an external reference.

7. The method of claim 1, wherein looking up the address for the desired element in the global reference table involves retrieving an absolute address for the desired element from the global reference table.

8. The method of claim 1, wherein looking up the address for the desired element in the global reference table involves retrieving a relative address for the desired element from the global reference table, wherein the relative address specifies an offset relative to a base address of a component containing the desired element.

9. The method of claim 1, wherein the desired element can include:

a class record;

a static field; and

a method.

10. The method of claim 1, wherein the reference is 16 bits in size and the address retrieved from the global reference table is 32 bits in size.

11. The method of claim 1, wherein the global reference table is stored as a multi-level table so that the entire global reference table does not have to be allocated at once.

12. The method of claim 1, wherein a given package includes:

a static field component containing static fields;

a class component containing class records;

a method component containing methods;

an export component; and

an import component.

13. A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for accessing a desired element during execution of a program, the method comprising:

receiving a reference to the desired element during execution of the program;

determining if the reference is an internal reference to a location in a local package that is currently executing, or an external reference to a location in an external package;

if the reference is an external reference, using an index component from the reference to lookup an address for the desired element in a global reference table; and

using the address to access the desired element;

wherein the address retrieved from the global reference table is larger than the reference, thereby allowing the address to access a larger address space than is possible to access with the reference alone.

14. The computer-readable storage medium of claim 13, wherein if the reference is an internal reference, the method further comprises:

determining a base address of a component in which the desired element is located; and

adding an offset of the reference to the base address to produce the address for the desired element.

15. The computer-readable storage medium of claim 14, wherein determining the base address of the component involves:

obtaining an identifier for the local package that is currently executing;

using the identifier to lookup an entry in a package location table, wherein the entry points to a component location table for the local package; and

examining the component location table to determine the base address of the component.

16. The computer-readable storage medium of claim 15, wherein examining the component location table involves first examining an instruction preceding the reference to determine which component the desired element is located in.

17. The computer-readable storage medium of claim 13, wherein the method further comprises installing a new package, wherein installing the new package involves:

calculating element addresses for all exported references associated with the new package;

adding the element addresses to the global reference table; and

replacing all external references within the new package with references to entries in the global reference table, wherein the entries in the global reference table contain element addresses for the desired elements.

18. The computer-readable storage medium of claim 13, wherein determining if the reference is an internal reference or an external reference involves examining a reserved bit within the reference, wherein the reserved bit indicates whether the reference is an internal reference or an external reference.

19. The computer-readable storage medium of claim 13, wherein looking up the address for the desired element in the global reference table involves retrieving an absolute address for the desired element from the global reference table.

20. The computer-readable storage medium of claim 13, wherein looking up the address for the desired element in the global reference table involves retrieving a relative address for the desired element from the global reference table, wherein the relative address specifies an offset relative to a base address of a component containing the desired element.

21. The computer-readable storage medium of claim 13, wherein the desired element can include:

a class record;

a static field; and

a method.

22. The computer-readable storage medium of claim 13, wherein the reference is 16 bits in size and the address retrieved from the global reference table is 32 bits in size.

23. The computer-readable storage medium of claim 13, wherein the global reference table is stored as a multi-level table so that the entire global reference table does not have to be allocated at once.

24. The computer-readable storage medium of claim 13, wherein a given package includes:

a static field component containing static fields;

a class component containing class records;

a method component containing methods;

an export component; and

an import component.

25. An apparatus that facilitates accessing a desired element during execution of a program, comprising:

an execution mechanism that is configured to receive a reference to the desired element during execution of the program; and

an address resolution mechanism that is configured to determine if the reference is an internal reference to a location in a local package that is currently executing, or an external reference to a location in an external package;

wherein if the reference is an external reference, the address resolution mechanism is configured to use an index component from the reference to lookup an address for the desired element in a global reference table;

wherein the execution mechanism is additionally configured to use the address to access the desired element;

wherein the address retrieved from the global reference table is larger than the reference, thereby allowing the address to access a larger address space than is possible to access with the reference alone.

26. The apparatus of claim 25, wherein if the reference is an internal reference, the address resolution mechanism is additionally configured to:

determine a base address of a component in which the desired element is located; and to

add an offset of the reference to the base address to produce the address for the desired element.

27. The apparatus of claim 26, wherein while determining the base address of the component, the address resolution mechanism is configured to:

obtain an identifier for the local package that is currently executing;

use the identifier to lookup an entry in a package location table, wherein the entry points to a component location table for the local package; and to

examine the component location table to determine the base address of the component.

28. The apparatus of claim 27, wherein while examining the component location table, the address resolution mechanism is configured to first examine an instruction preceding the reference to determine which component the desired element is located in.

29. The apparatus of claim 25, further comprising a package installation mechanism that is configured to:

receive a new package to be installed;

calculate element addresses for all exported references associated with the new package;

add the element addresses to the global reference table; and to

replace all external references within the new package with references to entries in the global reference table, wherein the entries in the global reference table contain element addresses for the desired elements.

30. The apparatus of claim 25, wherein while determining if the reference is an internal reference or an external reference, the address resolution mechanism is configured to examine a reserved bit within the reference, wherein the reserved bit indicates whether the reference is an internal reference or an external reference.

31. The apparatus of claim 25, wherein while looking up the address for the desired element in the global reference table, the address resolution mechanism is configured to retrieve an absolute address for the desired element from the global reference table.

32. The apparatus of claim 25, wherein while looking up the address for the desired element in the global reference table, the address resolution mechanism is configured to retrieve a relative address for the desired element from the global reference table, wherein the relative address specifies an offset relative to a base address of a component containing the desired element.

33. The apparatus of claim 25, wherein the desired element can include:
a class record;
a static field; and
a method.

34. The apparatus of claim 25, wherein the reference is 16 bits in size and the address retrieved from the global reference table is 32 bits in size.

35. The apparatus of claim 25, wherein the global reference table is stored as a multi-level table so that the entire global reference table does not have to be allocated at once.

36. The apparatus of claim 25, wherein a given package includes:

a static field component containing static fields;

a class component containing class records;

a method component containing methods;

an export component; and

an import component.

37. A means for accessing a desired element during execution of a program, comprising:

an execution means for receiving receive a reference to the desired element during execution of the program;

a determining means for determining if the reference is an internal reference to a location in a local package that is currently executing, or an external reference to a location in an external package;

a lookup means, wherein if the reference is an external reference, the lookup uses an index component from the reference to lookup an address for the desired element in a global reference table; and

wherein the execution means is additionally uses the address to access the desired element;

wherein the address retrieved from the global reference table is larger than the reference, thereby allowing the address to access a larger address space than is possible to access with the reference alone.

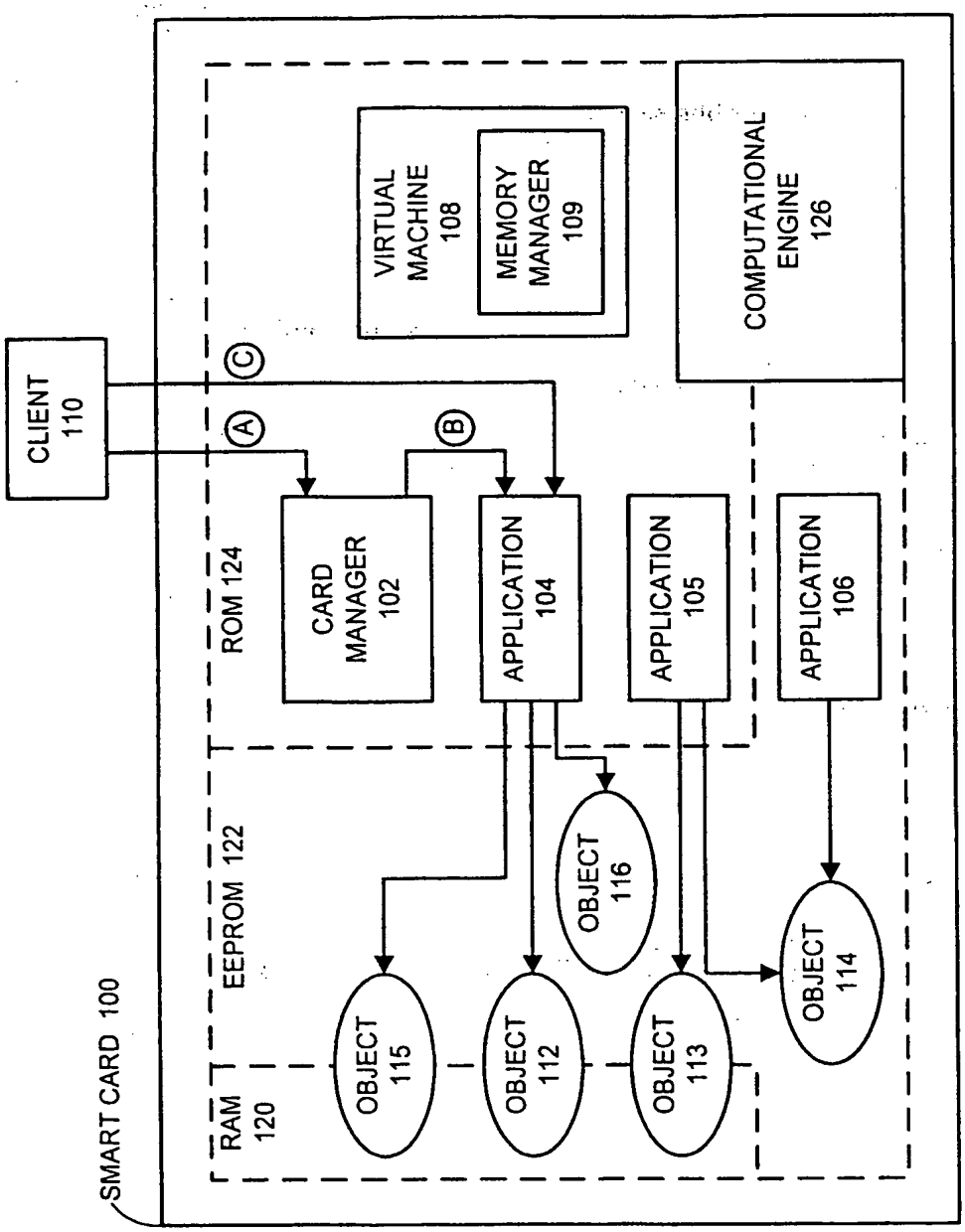

CLIENT 110
A
B
C
SMART CARD 100
RAM 120
EEPROM 122
ROM 124
VIRTUAL MACHINE 108
MEMORY MANAGER 109
COMPUTATIONAL ENGINE 126
CARD MANAGER 102
APPLICATION 104
APPLICATION 105
APPLICATION 106
OBJECT 115
OBJECT 112
OBJECT 116
OBJECT 113
OBJECT 114

FIG. 1

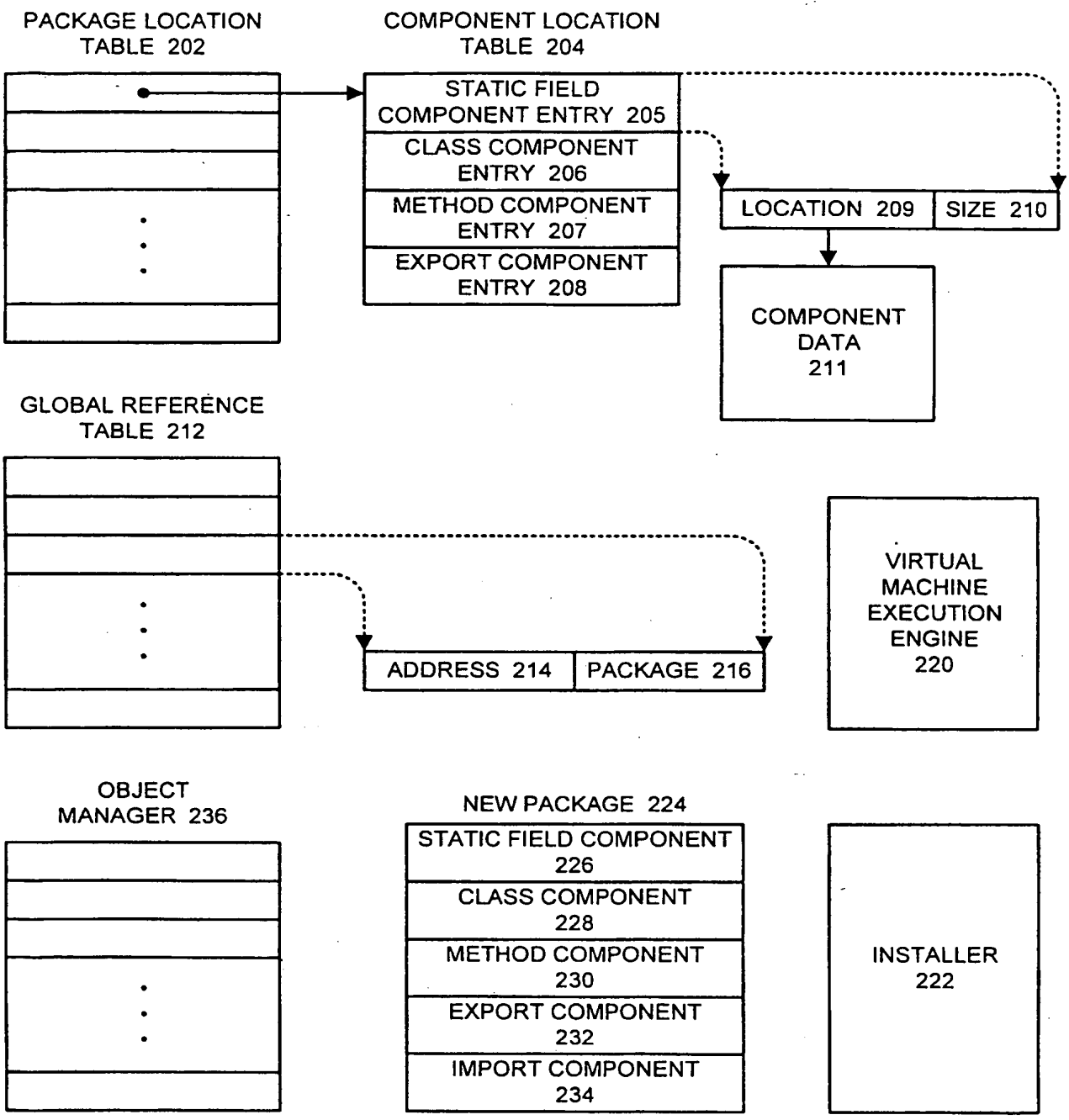

PACKAGE LOCATION TABLE 202
COMPONENT LOCATION TABLE 204
STATIC FIELD COMPONENT ENTRY 205
CLASS COMPONENT ENTRY 206
METHOD COMPONENT ENTRY 207
EXPORT COMPONENT ENTRY 208
LOCATION 209
SIZE 210
COMPONENT DATA 211
GLOBAL REFERENCE TABLE 212
ADDRESS 214
PACKAGE 216
VIRTUAL MACHINE EXECUTION ENGINE 220
OBJECT MANAGER 236
NEW PACKAGE 224
STATIC FIELD COMPONENT 226
CLASS COMPONENT 228
METHOD COMPONENT 230
EXPORT COMPONENT 232
IMPORT COMPONENT 234
INSTALLER 222

FIG. 2

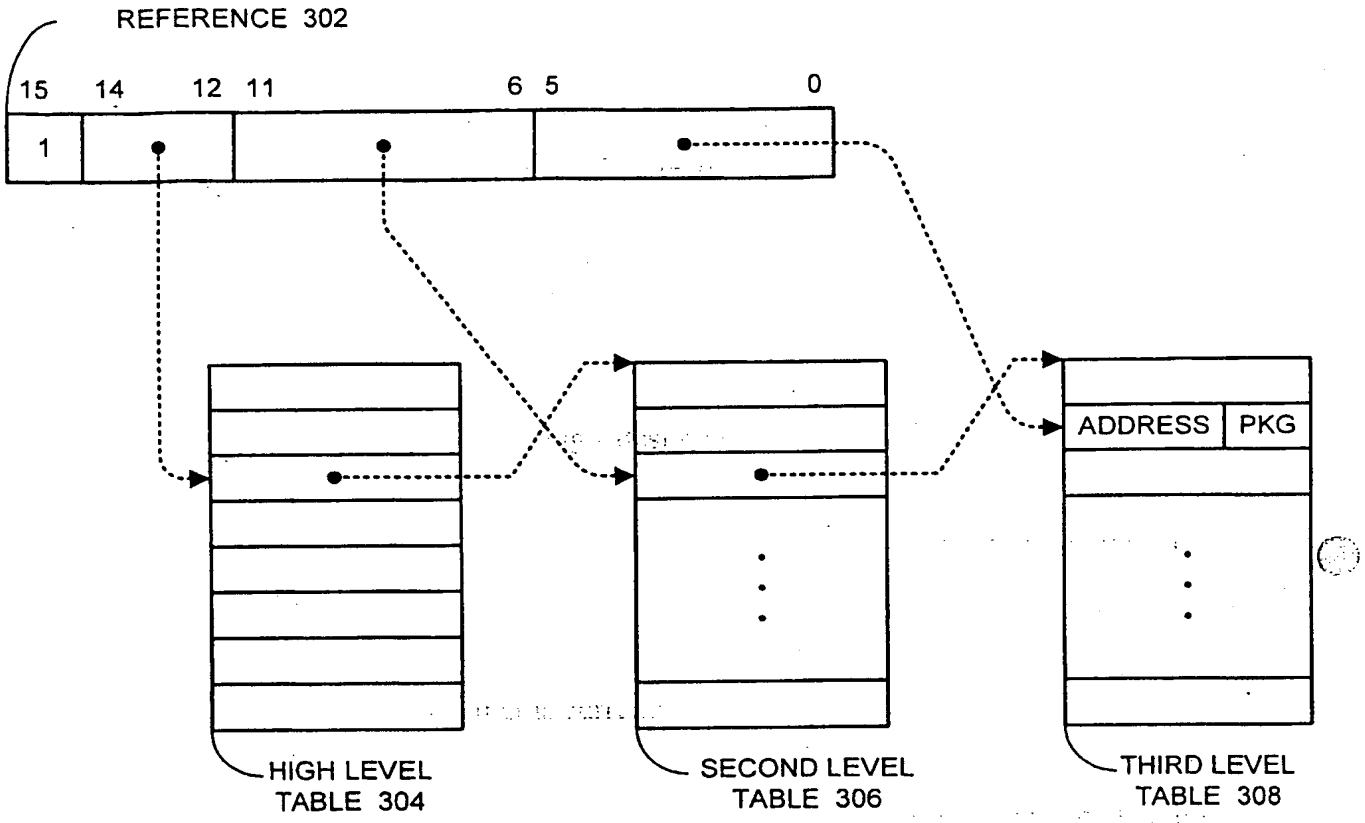

REFERENCE 302
15
14
12
11
6
5
0
1
ADDRESS
PKG
HIGH LEVEL TABLE 304
SECOND LEVEL TABLE 306
THIRD LEVEL TABLE 308

FIG. 3

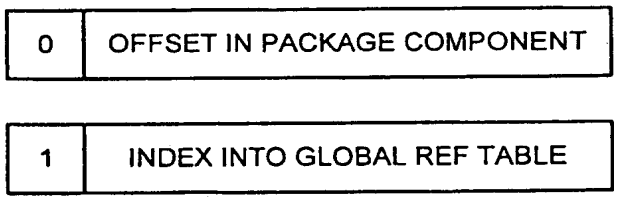

0
OFFSET IN PACKAGE COMPONENT
1
INDEX INTO GLOBAL REF TABLE

FIG. 4

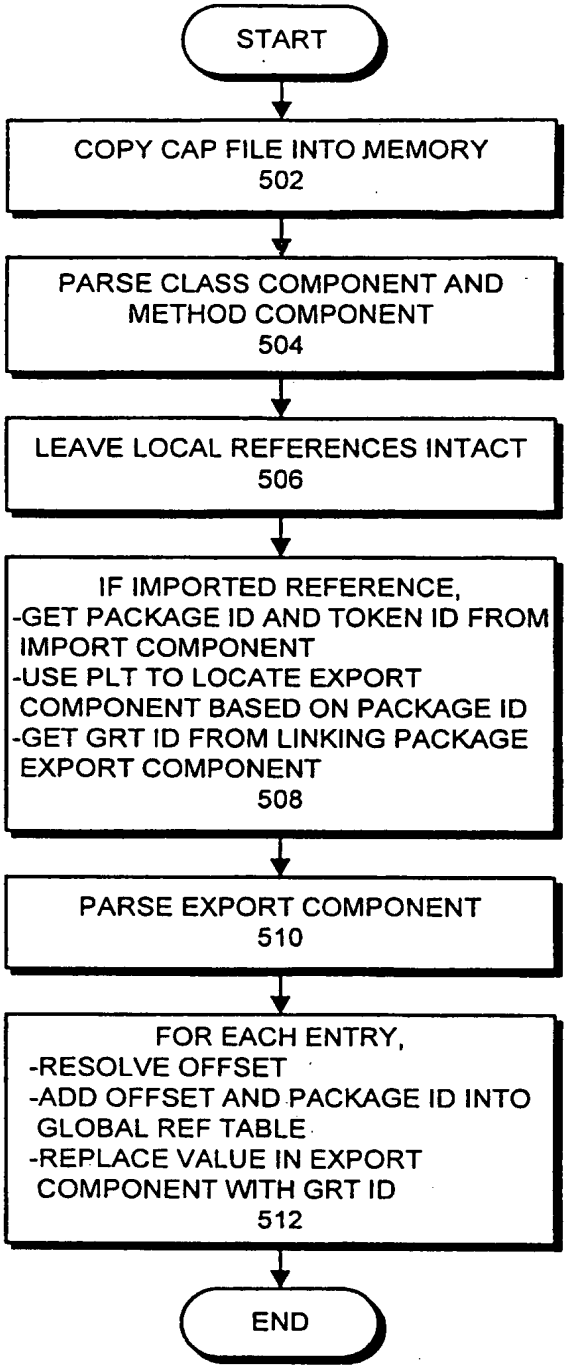


FIG. 5

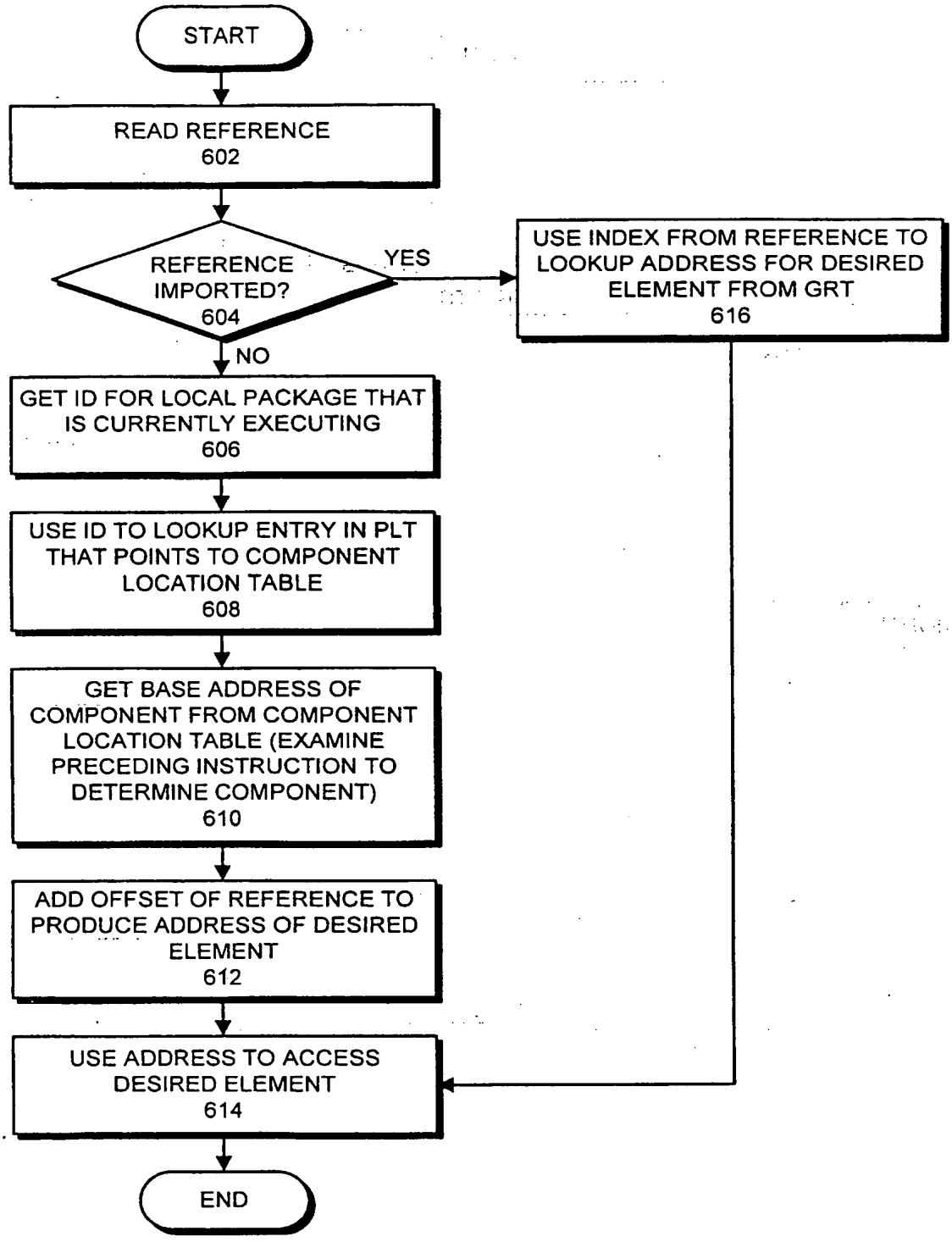

START
READ REFERENCE
602
REFERENCE IMPORTED?
604
YES
USE INDEX FROM REFERENCE TO LOOKUP ADDRESS FOR DESIRED ELEMENT FROM GRT
616
NO
GET ID FOR LOCAL PACKAGE THAT IS CURRENTLY EXECUTING
606
USE ID TO LOOKUP ENTRY IN PLT THAT POINTS TO COMPONENT LOCATION TABLE
608
GET BASE ADDRESS OF COMPONENT FROM COMPONENT LOCATION TABLE (EXAMINE PRECEDING INSTRUCTION TO DETERMINE COMPONENT)
610
ADD OFFSET OF REFERENCE TO PRODUCE ADDRESS OF DESIRED ELEMENT
612
USE ADDRESS TO ACCESS DESIRED ELEMENT
614
END

FIG. 6

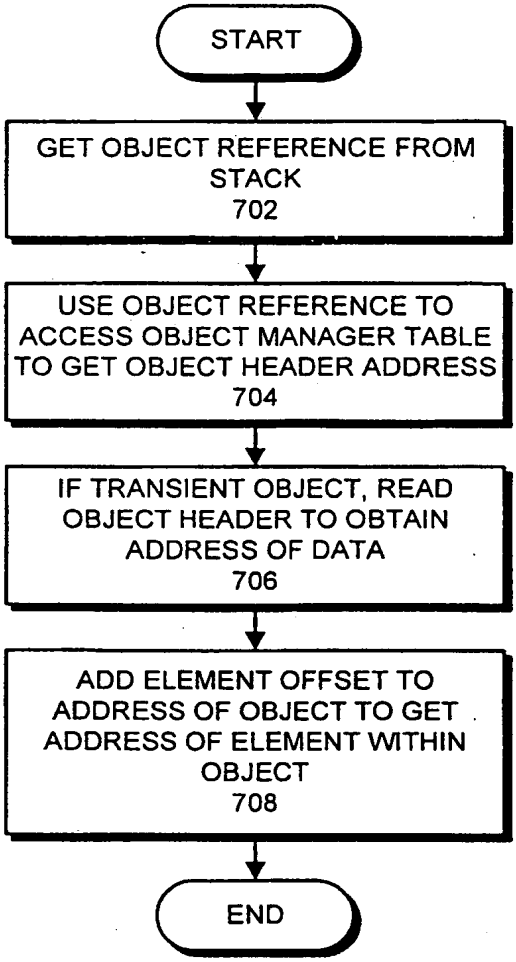


FIG. 7

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 December 2003 (18.12.2003)

PCT

(10) International Publication Number
WO 2003/104974 A3

(51) International Patent Classification[7]: G06F 9/455, 9/46

(21) International Application Number:
PCT/US2003/011419

(22) International Filing Date: 11 April 2003 (11.04.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/386,856	7 June 2002 (07.06.2002)	US
10/165,160	7 June 2002 (07.06.2002)	US

(71) Applicant: SUN MICROSYSTEMS, INC. [US/US]; 4150 Network Circle, Santa Clara, CA 95054 (US).

(72) Inventor: MONTEMAYOR, Oscar; 1035 Aster Ave. Apt. 1134, Sunnyvale, CA 94086 (US).

(74) Agent: PARK, A., Richard; Park, Vaughan & Fleming LLP, 2820 Fifth Street, Davis, CA 95616 (US).

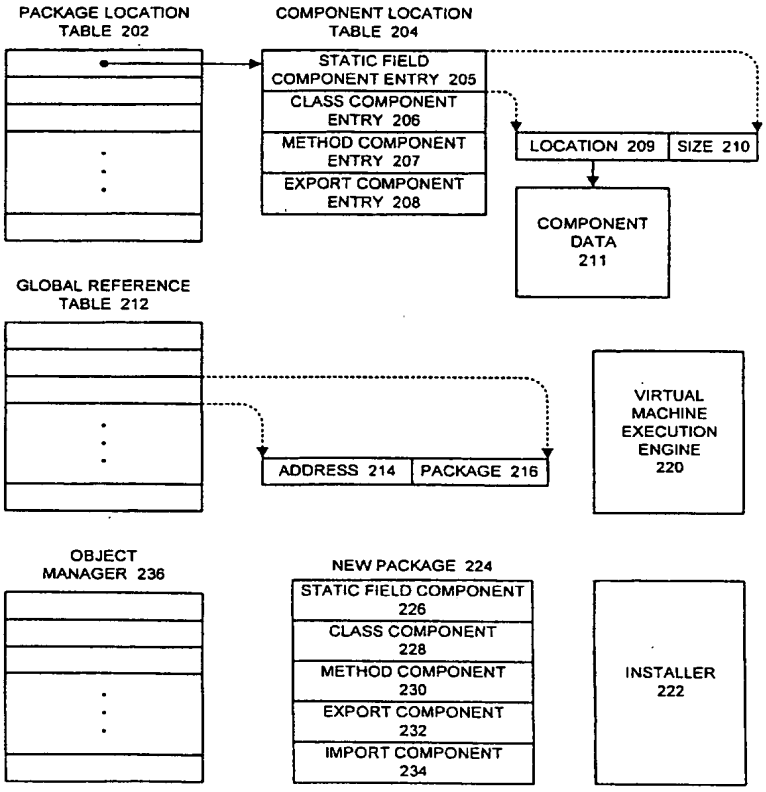
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NI, NO, NZ, OM, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

[Continued on next page]

(54) Title: USING SHORT REFERENCES TO ACCESS PROGRAM ELEMENTS IN A LARGE ADDRESS SPACE



(57) Abstract: One embodiment of the present invention provides a system that accesses a desired element during execution of a program. During operation, the system receives a reference to the desired element. The system determines if the reference is an internal reference to a location in a local package that is currently executing, or an external reference to a location in an external package. If the reference is an external reference, the system uses an index component of the reference to lookup an address for the desired element in a global reference table. Next, the system uses the address to access the desired element. Note that the address retrieved from the global reference table is larger than the reference, which allows the address to access a larger address space than is possible to access with the reference alone.

WO 2003/104974 A3

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) Date of publication of the international search report:
2 June 2005

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

Intern Application No
PCT/US 03/11419

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/455 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	MAURICE J. BACH: "The Design of the UNIX Operating System" 1990, PRENTICE HALL , USA , XP002323844 ISBN: 0-13-201799-7 page 285 - page 307,Section 9.2 "Demand Paging"	1,2, 5-14, 17-26, 29-37
A	DAVID A. PATTERSON & JOHN L. HENNESSY: "Computer Architecture -- A Quantitative Approach" 1996, MORGAN KAUFMANN PUBLISHERS , SAN FRANCISCO , XP002323845 ISBN: 1-55860-329-8 page 439 - page 453, Sections 5.7 and 5.8	2,7,8, 11,14, 19,20, 23,26, 31,32,35

-/--

[X] Further documents are listed in the continuation of box C. [ ] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
11 April 2005	03/05/2005

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Müller, T

Form PCT/ISA/210 (second sheet) (January 2004)

INTERNATIONAL SEARCH REPORT

Inter I Application No
PCT/US 03/11419

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	JOHN R. LEVINE: "Linkers & Loaders" 2000, MORGAN KAUFMANN PUBLISHERS , SAN FRANCISCO , XP002323846 ISBN: 1-55860-496-0 page 118 - page 124, Section 5.2 "Symbol Table Formats" page 241 - page 244, Section 11.6 "The JAVA Linking Model"	5,17,29
A	PARHAMI B ED - SODERSTRAND M A ET AL: "Modular reduction by multi-level table lookup" CIRCUITS AND SYSTEMS, 1997. PROCEEDINGS OF THE 40TH MIDWEST SYMPOSIUM ON SACRAMENTO, CA, USA 3-6 AUG. 1997, NEW YORK, NY, USA,IEEE, US, vol. 1, 3 August 1997 (1997-08-03), pages 381-384, XP010272518 ISBN: 0-7803-3694-1 the whole document	11,23,35

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

Continuation of Box I.2

Claims Nos.: 3,4,15,16,27,28

A lack of clarity within the meaning of Article 6 PCT arises to such an extent as to render a meaningful search of the claims impossible. Consequently, the search has been carried out for those parts of the application which do appear to be clear, namely claims 1, 2, 5 to 14, 17 to 26 and 29 to 37. Claims 3, 4, 15, 16, 27 and 28 contain the incomplete phrase "wherein the entry points to a component location table for the local package;" which renders a meaningful search for those claims impossible.

The applicant's attention is drawn to the fact that claims relating to inventions in respect of which no international search report has been established need not be the subject of an international preliminary examination (Rule 66.1(e) PCT). The applicant is advised that the EPO policy when acting as an International Preliminary Examining Authority is normally not to carry out a preliminary examination on matter which has not been searched. This is the case irrespective of whether or not the claims are amended following receipt of the search report or during any Chapter II procedure. If the application proceeds into the regional phase before the EPO, the applicant is reminded that a search may be carried out during examination before the EPO (see EPO Guideline C-VI, 8.5), should the problems which led to the Article 17(2) declaration be overcome.

INTERNATIONAL SEARCH REPORT

Ir ɔnal application No.
PCT/US 03/11419

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☒ Claims Nos.: 3,4,15,16,27,28
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:
see FURTHER INFORMATION sheet PCT/ISA/210

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.